LECTURE 5

Master theorem

A **recurrence equation** (or simply **recurrence**) arises when we split a problem of size n into smaller problems, solve the smaller problems recursively, and then re-combine the solutions of the smaller problems. Ex. Merge-sort.

In other words, recurrences arise in the context of **divide and conquer** algorithms. The following theorem is applicable in such cases.

Theorem 4.1 (Master theorem) [CLRS]

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on non-negative integers by the recurrence:

$T(n) = aT(n/b) + f(n)$ [see diagram on next page]

where we may interpret n/b to mean either floor(n/b) or ceiling(n/b). Then T(n) can be bounded asymptotically as follows:

Case 1: $f(n) = O(\ n^{(\log_b a - \varepsilon)}\ )$ for some constant $\varepsilon > 0$

Then $T(n) = \Theta(n^{(\log_b a)})$

Case 2: $f(n) = O(n^{(\log_b a)})$ { Note: big O → upper bound }

Then $T(n) = \Theta(n^{(\log_b a)} \lg n)$

Case 3: $f(n) = O(\ n^{(\log_b a + \varepsilon)}\ )$ for some constant $\varepsilon > 0$ &&
$a*f(n/b) \leq c*f(n)$ for some constant $c < 1$ for all sufficiently large n

Then $T(n) = \Theta(\ f(n)\ )$

What is the significance of a, b and f(n)?

Does the master theorem apply to **merge-sort**?

Let us take a closer look at the above recurrence:

T(n) = aT(n/b) + f(n)

Cost of recursive function calls

Cost of splitting & combining

Case 1: The cost of recursive function calls dominates (asymptotically).

Case 2: Neither cost dominates the other (asymptotically).

Case 3: The cost of splitting & combining dominates (asymptotically).

Points to note:

1. What do we do if n is <u>not</u> a multiple of b?

Answer: Ignore the fractional parts, taking ceiling or floor ... Recall that we are looking only for asymptotic bounds.

2. What is the so-called "boundary condition" – the point at which we do not split the problem into sub-problems?

Generally, T(n) = Θ (1) for "small" n. This provides the required boundary condition → no more recursion; we are at a leaf node of the recursion tree. For getting the asymptotic Θ bound, we should overlook the boundary condition.

3. Sometimes, the <u>substitution method</u> or the <u>recursion tree method</u> may prove useful; but of course the master theorem gives the general solution for such recurrences.

[Recall that the substitution method can be tried with any system of equations, provided only that we are able to think of a reasonable trial solution.]

4. Running times of **iterative algorithms** can often be found by using **induction**. Recall how the running time of insertion sort was determined.